J. Symbolic Computation (1991) **12**, 397–414

The Use of Knuth–Bendix Methods to Solve the Word Problem in Automatic Groups

D. B. A. EPSTEIN, D. F. HOLT AND S. E. REES

Mathematics Institute, University of Warwick, Coventry CV4 7AL, UK

(*Received* 18 *May* 1989)

Certain classes of infinite groups arising from geometry and topology are known to have solvable word problem. We describe the development of practical methods for the solution of the word problem based on the reduction of words in the generators to a normal form. The Knuth–Bendix completion procedure is the principal tool used but, in the case that this process does not halt, we use alternative methods involving the construction of finite-state automata. A computer implementation of these procedures together with some performance statistics on some simple examples are also described.

1. Introduction

A famous result, due independently to Boone and Novikov, asserts that the word problem is unsolvable for groups in general. In fact one could, in principle at least, write down a specific group presentation, with finitely many generators and relations, for which there can be no mechanical decision procedure to determine whether a given word in the generators represents the identity element in the group. However, groups which turn up in practice in many areas of mathematics, including geometry and topology, frequently turn out to have solvable word problem. Unfortunately, many of the currently known results in this area merely assert the existence of a decision procedure for the relevant groups, without telling us how to do it in practice. (For example, the proof of the solvability of the word problem for certain classes of knot groups involves the step: Consider all planar diagrams with at most n^2 vertices, where n is the length of the word in question. It is presumably clear to the reader that such a process would always be totally impractical, even on the fastest of computers.) On the other hand, with the rapidly increasing speed and user-friendliness of electronic computers, in conjunction with more and more instances of their successful application to problems in pure mathematics, there is a growing demand from the geometers and topologists for the development of practical procedures for carrying out elementary computations within certain infinite presented groups. Such procedures would certainly find immediate applications to the study of the underlying geometrical and topological objects. (For finite groups, the position is much rosier. The main computational tool here is Todd–Coxeter coset enumeration, which was one of the first substantial computer programs of any kind ever to be written, and is currently in an advanced state of development.)

Practical methods of solving the word problem in algebraic systems in general usually require some sort of normal form for the elements of the system, together with an efficient algorithm for transforming an arbitrary word in the generators into this normal form. One such method is to find a *complete reduction system.* This consists, roughly speaking,

0747-7171/91/100397+18 $03.00/0

of a system of reductions (or directed equations) of the form $v \to w$ with the following property. Any word that is not already in normal form contains the left-hand side of one of these equations as a subword, and any such word can be reduced to a uniquely determined element in normal form, by a finite number of substitutions of the left-hand side of one of the equations by its right-hand side. Such a system of equations is called *complete* or *confluent*. For finite systems of equations, a special case of the Knuth-Bendix Lemma provides an efficient method of testing for confluence and, in the case of failure, for extending the system. For example, for the infinite abelian group $\langle a, b \mid ba = ab \rangle$, the system of reductions

$$a^{-1}a \to \varepsilon, \quad aa^{-1} \to \varepsilon, \quad b^{-1}b \to \varepsilon, \quad bb^{-1} \to \varepsilon,$$
$$ba \to ab, \quad ba^{-1} \to a^{-1}b, \quad b^{-1}a \to ab^{-1}, \quad b^{-1}a^{-1} \to a^{-1}b^{-1}$$

is confluent (where ε denotes the empty string, which corresponds to the identity element of the group), and can be used to reduce any word to the normal form $a^m b^n$, for $m, n \in \mathbb{Z}$.

A good description of the use of Knuth-Bendix methods for solving the word problem in groups and monoids can be found in the papers of Gilman (1979, 1985), and several examples of infinite groups with finite confluent systems are listed in Le Chenadec (1986). The main problem, however, is that for most presentations of infinite groups, this method fails to generate a finite system, but continues indefinitely, and generates an infinite confluent system. It was observed by Gilman (1985) that, in many important cases, this infinite system is *regular* in a technical sense, which implies in particular that it can be finitely described, and it is almost as efficient as a finite system for reducing words to normal form. Indeed, in some cases it is possible to achieve a finite confluent system by a slight modification of the presentation of the group, or of the normal form. This does not seem to be possible in general, however, and there is certainly no systematic method of achieving it. The principal difficulty inherent in these regular infinite confluent systems is the difficulty of verifying confluence. In fact it is shown in Benninghofen *et al.* (1987, chapter III.1.3) that the confluence of such a system is an undecidable property in general.

The purpose of this paper is to describe an implementation of an effective solution to this problem for certain *automatic* groups. This class of groups includes many of the examples mentioned above that arise naturally in geometry and topology (for example, hyperbolic and Euclidean groups). Five of the people most closely involved in this work are currently collaborating in the production of a detailed and substantial document on automatic groups (Cannon *et al.*, preprint). Here, we shall develop only as much of this theory as we need to keep this paper reasonably self-contained.

The approach indicated by the theory of automatic groups is not to verify confluence directly, or even to describe the complete reduction system. Instead, we run the Knuth-Bendix algorithm until we believe that we have generated enough equations to capture the essence of the complete system. We then construct a finite state automaton W which is intended to accept precisely the reduced words (that is, the words in normal form). (As an important by-product, it can also be used effectively to reduce arbitrary words to normal form.) We then construct other (unfortunately much larger) finite state automata which can be used to prove the *correctness* of W; that is, we can verify that W has the intended property. In case of failure, we have to go back and run the Knuth-Bendix algorithm a bit longer.

The paper is organized as follows. In section 2, we define and develop the required properties of automatic groups. In section 3, we describe our implementation of the Knuth-Bendix algorithm, and in section 4 we explain how the output is used to construct

the relevant finite state automata. Then, in section 5, we show how these are used to verify correctness of W. Finally, in section 6, we provide some experimental data on some examples.

2. Automatic Groups

Throughout this paper, we shall be dealing with finite presentations $\langle A \mid R \rangle$ of groups G (A is for *alphabet*). It will be convenient to assume that the generating set A is always closed under inversion. In other words, for each $a \in A$, there exists $a' \in A$ (possibly equal to a) such that aa' and $a'a$ lie in R, and we assume also that $a'' = a$. This means that we really have a presentation of G as a monoid. If w is a word in A, then we shall denote its image in G by $\bar{w}$.

Let A^* be the set of words (or strings) over A, including the empty string ε. It would be convenient for us if we could regard elements (v, w) of $A^* \times A^*$ as elements of $(A \times A)^*$. This is technically not possible, since v and w will not always have the same length. We get round this problem by introducing an extra symbol \$, which is assumed not to lie in A, and which maps onto the identity in G, and, if necessary, we append \$ to the end of the shorter of the two words until they have equal length. We shall write $(v^\dagger, w^\dagger)$ or simply $(v, w)^\dagger$ for the resulting element of $(A^\dagger \times A^\dagger)^*$, where $A^\dagger = A \cup \{\$\}$, and we refer to an element of $(A^\dagger \times A^\dagger)^*$ that arises in this way as a *padded* word over $A \times A$.

We can now define an automatic group. We shall assume that the reader is familiar with the definition and elementary properties of a finite state automaton (see, for example, Hopcroft & Ullman (1979), or any book on formal language theory).

DEFINITION 2.1. The group G is said to be *automatic* with respect to the generating set A if there exist finite state automata W, $M_\$$, and M_a for each $a \in A$, with the following properties.

(i) W has input alphabet A and, for each $g \in G$, there is at least one element $w \in A^*$ with $w \in L(W)$ (the language accepted by W) and $\bar{w} = g$.

(ii) For $a = \$$ or $a \in A$, M_a has input alphabet $A^\dagger \times A^\dagger$, M_a accepts only padded words over $A \times A$ and, for all $v, w \in A^*$, $(v, w)^\dagger \in L(M_a)$ if and only if $v, w \in L(W)$ and $\bar{v}\bar{a} = \bar{w}$.

This definition was originally suggested by W. P. Thurston, as a reinterpretation of results of J. W. Cannon. Cannon's work (1984) is an investigation of properties of groups of isometries acting discretely and cocompactly on hyperbolic space.

In order to give the reader a feel for the type of groups under discussion, we shall now summarize some of the most important properties and examples of automatic groups that have been proved to date. Proofs can be found in Cannon (1984).

The property of being automatic is in fact independent of the choice of generating set A. All automatic groups are finitely presented, and have solvable word problem. The class of automatic groups is closed under direct and free products, free products with a finite amalgamated subgroups, HNN extensions with finite conjugated subgroup, subgroups and supergroups of finite index, and quotient groups by a normal subgroup of finite order. They are clearly not closed under quotients in general, and also not under subgroups, since subgroups in general need not be finitely generated.

In addition to the hyperbolic groups mentioned above, free groups and free abelian groups are automatic, and also groups satisfying some of the standard small-cancellation conditions. All finite groups are of course automatic, but this is unlikely to provide an

efficient approach to computation in finite groups (which is already in a very advanced state of development), except possibly in a few isolated cases.

Every torsion-free non-abelian nilpotent group is not automatic; indeed, we have now just about convinced ourselves that "nilpotent" can be replaced by "polycyclic" in this statement.

The following definition and theorem will form the basis of our algorithm for constructing W and the M_a. From now on, we shall assume that all finite state automata are deterministic.

DEFINITION 2.2. Let $(u^\dagger, v^\dagger)$ be a padded word over $A \times A$, and let $u^\dagger = a_1 a_2 \dots a_n$ and $v^\dagger = b_1 b_2 \dots b_n$. Then the subset $\{(\bar{a}_1 \bar{a}_2 \dots \bar{a}_m)^{-1}(\bar{b}_1 \bar{b}_2 \dots \bar{b}_m) \mid 0 \le m \le n\}$ of G is called the set of *word-differences* of (u, v). If E is a set of padded words over $A \times A$, then the union of the sets of word differences for all $(u^\dagger, v^\dagger) \in E$ is called the set of word differences of E.

THEOREM 2.3. *Let G be an automatic group. Then, for each $a \in A$ and for $a = \$$, the set of word differences of $L(M_a)$ is finite.*

PROOF. Since M_a has finitely many states, there is a constant d with the following property: if s is any state of M_a that can lead to success, then there is a padded word over $A \times A$ of length at most d that leads to success when input to M_a in state s. Now, for each word difference g of $L(M_a)$, we have $g = (\bar{b}_1 \bar{b}_2 \dots \bar{b}_m)^{-1}(\bar{c}_1 \bar{c}_2 \dots \bar{c}_m)$, where M_a is mapped, from its initial state, into state s by $(b_1 b_2 \dots b_m, c_1 c_2 \dots c_m)$, and s can lead to success. Hence there exists $(b_{m+1} \dots b_n, c_{m+1} \dots c_n)$ with $n - m \le d$ that maps M_a to a success state from state s. By definition of M_a this implies that $\bar{b}_1 \bar{b}_2 \dots \bar{b}_n \bar{a} = \bar{c}_1 \bar{c}_2 \dots \bar{c}_n$, and so $g = (\bar{b}_{m+1} \bar{b}_{m+2} \dots \bar{b}_n \bar{a})(\bar{c}_{m+1} \bar{c}_{m+2} \dots \bar{c}_n)^{-1}$ is the image of a word in A of length at most $2d + 1$. Since there are only finitely many such words, the result follows.

This theorem provides an approach to a possible construction of the M_a in practice. Since both components of words accepted by M_a must be accepted by W, M_a can consist of three independent automata. Two of these are copies of W, and test the acceptability of the two components, and the third keeps track of the word difference between the two components, and rejects the input if this word difference goes outside of the appropriate finite set. If $g \in G$ is a word difference and $x, y \in A$, then the effect of (x, y) on g is to give $\bar{x}^{-1} g \bar{y}$, as we see from Definition 2.2. (More precise and more accurate details are given in the statement of Corollary 2.4.) The input will only be accepted by a particular M_a if the final word difference is equal to $\bar{a}$.

There is an additional technical problem caused by the fact that one of the two components of the input may end with one or more copies of the padding symbol \$. We deal with this as follows. We assume that W has at least one failure state (i.e. one that is mapped to itself by any input), which we shall denote by 0_W. We now introduce an extra accept state ∞ to the set of states of W, such that the accept states of W are mapped by \$ into the state ∞, whereas the non-accept states are mapped by \$ to 0_W.

COROLLARY 2.4. Let G, W and M_a be as in the theorem, and let D be the set of word differences of $L(M_a)$. Then the following finite state automaton F is equivalent to M_a (that is, it defines the same language as M_a). The set of states of F is equal to $(D \times (S \cup \{\infty\}) \times (S \cup \{\infty\})) \cup \{0_F\}$, where S is the set of states of W (including a failure state 0_W). The initial state is $(1, s_0, s_0)$, where 1 is the identity element of G and s_0 is the

initial state of W. For states $s \in S$, we denote the image of s under $b \in A$ in W by s^b, and also define $s^\$ = \infty^\$ = \infty$ for accept states s and $s^\$ = 0_W$ for non-accept states s. Then the state 0_F is mapped to 0_F by any input, and element $(d, s, t) \in D \times (S \cup \{\infty\}) \times (S \cup \{\infty\})$ is mapped by $(b, c) \in A^\dagger \times A^\dagger$ onto 0_F if $\bar{b}^{-1} d \bar{c} \notin D$, or if $b = c = \$$, or if $b \in A$ and $s = \infty$ or if $c \in A$ and $t = \infty$, and onto $(\bar{b}^{-1} d \bar{c}, s^b, t^c)$ otherwise. The success states are those of the form $(\bar{a}, s, t)$, where s and t are equal either to ∞ or to success states of W.

PROOF. It is easy to see that F fulfils the conditions for M_a that are specified in (ii) of Definition 2.1. (Note that there may be many inaccessible states of F. For example, (d, ∞, ∞) is not accessible for any d.)

Our programs which attempt to construct W and the M_a, and which will be described in section 4, will in fact attempt to follow the recipe described in this corollary. The only difference will be that we shall not be in a position to compute the elements of D as elements of G. We can only compute them as elements of A^* that map onto elements of G.

In this paper, we shall in fact restrict our attention to groups G which are *shortest word automatic* with respect to an *ordered* generating set A, in the sense that, for each group element g, W accepts only the lexicographically least amongst the words w of shortest length that satisfy $\bar{w} = g$. Such a word w is called the *normal form* for g. In contradistinction to automatic groups in general, this property definitely does depend on the choice of A rather than only on G, and it could conceivably depend on the chosen ordering of A. However, many groups of interest, such as hyperbolic groups and abelian groups, are shortest word automatic with respect to *any* choice of generators. Also, certain small-cancellation hypotheses, such as $C'(1/6)$, or $C'(1/4)$ and $T(4)$, which involve a choice of generators and relators as part of their definition, also imply the shortest word automatic property. The groups in the class recently investigated by Gromov (1987) are also shortest word automatic with respect to any choice of generators. Under this restricted hypothesis, the automaton $M_\$$ is easily constructed from W, since $(v, w) \in L(M_\$)$ if and only if $v = w \in L(W)$.

Our principal aim will be to construct the automaton W since, as we shall see in sections 3 and 4, our construction will in fact provide us with enough additional information to reduce any word in A to normal form. The M_a are not in themselves particularly useful for solving the word problem, since they do not tell us how to multiply by a generator; they only tell us when we have guessed the right answer. However, in most cases, we need to construct the M_a in order to prove the correctness of our candidate for W. The algorithm that we employ for this purpose is based on the following theorem. A version of this is proved for automatic groups in general in Chapter 10 of Cannon *et al.* (preprint). We include its proof here mainly in order to make this current paper self-contained, but also because its proof is more straightforward under our more restrictive hypotheses.

THEOREM 2.5. *Let $G = \langle A | R \rangle$ be a finitely presented group, and let W and M_a (for $a \in A$) be finite state automata satisfying the following conditions.*

(i) *W has input alphabet A, and each M_a has input alphabet $A^\dagger \times A^\dagger$ and accepts only padded words in $A \times A$.*

(ii) *If $(v, w)^\dagger \in L(M_a)$ for some $a \in A$, then $v, w \in L(W)$ and $\bar{v}\bar{a} = \bar{w}$ in G.*

(iii) *$L(W)$ is non-empty and, if $a_1 a_2 \ldots a_n \in L(W)$ for any $n > 0$, then $a_1 a_2 \ldots a_{n-1} \in L(W)$ and $(a_1 a_2 \ldots a_{n-1}, a_1 a_2 \ldots a_n)^\dagger \in L(M_{a_n})$.*

(iv) *Let* $w(=w_0)\in L(W)$ *and let* $a_1a_2\ldots a_n\in R$. Then, for any $w_n\in L(W)$, *there exist elements* $w_1, w_2, \ldots, w_{n-1}\in L(W)$ *with* $(w_{i-1}, w_i)^\dagger\in L(M_{a_i})$ *for* $1\leq i\leq n$ *if and only if* $w=w_n$.

Then G is automatic using these automata and, for each group element $g\in G$, *W accepts a unique word mapping onto g.*

PROOF. Let $a\in A$ and let $w\in L(W)$. Then, by our general hypothesis, there exists $a'\in A$ such that $aa', a'a\in R$. By applying hypothesis (iv) to w and aa', we deduce that there exists $w'\in L(W)$ such that $(w, w')^\dagger\in L(M_a)$ and $(w', w)^\dagger\in L(M_{a'})$. By applying it to w' and $a'a$, we find that w' is unique. Hence the map $\varphi(a): L(W)\to L(W)$ which maps w to w' is a permutation of $L(W)$. Furthermore, hypothesis (iv) implies that for any $a_1a_2\ldots a_n\in R$ the permutation $\varphi(a_1)\varphi(a_2)\ldots\varphi(a_n)$ of $L(W)$ is equal to the identity, and so the map φ extends to a homomorphism from G to the symmetric group on $L(W)$. Here we are thinking of the symmetric group as acting on $L(W)$ from the right. (Note that, since $aa'\in R$, we have $\varphi(a)\varphi(a')=1$ and so $\varphi(a)^{-1}=\varphi(a')$.)

Now let $g\in G$, and let $a_1a_2\ldots a_n$ be a word in A that maps onto g. Note that the empty string ε lies in $L(W)$ by hypothesis (iii), and it follows from hypothesis (ii) that the image of ε in $L(W)$ under the permutation $\varphi(a_1)\varphi(a_2)\ldots\varphi(a_n)$ also maps onto g. Therefore $L(W)$ contains at least one element mapping onto g, which we may as well assume to be $a_1a_2\ldots a_n$. Now let $b_1b_2\ldots b_m$ be any word in $L(W)$ that maps onto g. Then, since φ is a homomorphism defined on the group G, we have

$$\varphi(a_1)\varphi(a_2)\ldots\varphi(a_n)=\varphi(a_1a_2\ldots a_n)=\varphi(b_1b_2\ldots b_m)=\varphi(b_1)\varphi(b_2)\ldots\varphi(b_m).$$

However, it follows from hypothesis (iii) that

$$\varphi(a_1)\varphi(a_2)\ldots\varphi(a_n)\quad\text{and}\quad\varphi(b_1)\varphi(b_2)\ldots\varphi(b_m)$$

map ε to $a_1a_2\ldots a_n$ and $b_1b_2\ldots b_m$, respectively, and so the words $a_1a_2\ldots a_n$ and $b_1b_2\ldots b_m$ are in fact equal in A^* (not merely equal in G).

We have now proved that W accepts a unique word for each element of G. It follows that, if $v, w\in L(W)$ and $\bar{v}\bar{a}=\bar{w}$, then $(v, w)\in L(M_a)$ because, for a given $v\in L(W)$, w must be the unique word in $L(W)$ with $\bar{v}\bar{a}=\bar{w}$. This completes the proof of the theorem.

In fact, the automata that we construct, which will be described in section 4, will automatically satisfy hypotheses (i)–(iii), and so only hypothesis (iv) will need to be verified. This is done by constructing and analysing other automata, as will be described in section 5.

3. The Knuth–Bendix Algorithm

As in the previous section, we assume that the group G is generated by the finite ordered set A, and that A is closed under inversion. Let E (for *equations*) be a set of ordered pairs (v, w) with $v, w\in A^*$, with the property that the equations $v=w$, for $(v, w)\in E$, form a set of defining relations for G as a monoid. (We shall generally refer to an ordered pair (v, w) with $v, w\in A^*$, for which $\bar{v}=\bar{w}$ in G, as an *equation.*)

For $v, w\in A^*$, we define $v>w$ if either v is longer than w, or v and w have the same length but v comes after w in the dictionary ordering relative to the specified ordering on A. (More generally, any total ordering on A^* that satisfies the descending chain condition and the property that $v>w$ implies $uv>uw$ and $vu>wu$ for all $u\in A^*$ can be used, but we shall restrict our attention to this particular ordering, which has probably

been used the most frequently. Note that, for all $g \in G$, we want W to accept the least word under $<$ that represents g.) We may assume that $v > w$ for all $(v, w) \in E$. For $u, v \in A^*$, we write $u \to v$ if there exist $w, x, y, z \in A^*$ such that $u = wxz$, $v = wyz$ and $(x, y) \in E$. Let $\to^+$ denote the transitive closure of $\to$, $\to^*$ the reflexive closure of $\to^+$ and $\leftrightarrow^*$ the reflexive symmetric transitive closure of $\to^*$. Then G is isomorphic as a monoid to the factor monoid $A^*/\leftrightarrow^*$. We want to construct W such that $L(W)$ consists precisely of the least elements under $<$ in each equivalence class under $\leftrightarrow^*$. We shall say that a finite state automaton W with input alphabet A is *correct* if it has this property.

An element $u \in A^*$ is called *irreducible* under E if there does not exist t in A^* with $u \to t$. The set E is called confluent if, whenever $u, v, w \in A^*$, $u \to^* v$ and $u \to^* w$, then there exists $t \in A^*$ with $v \to^* t$ and $w \to^* t$. It is not difficult to show that E is confluent if and only if, for each $u \in A^*$, either u is irreducible, or there exists a unique irreducible $t \in A^*$ with $u \to^+ t$. Furthermore, if E is confluent, then each equivalence class under $\leftrightarrow^*$ contains a unique irreducible element, namely its least element under $<$. This means that we can easily reduce any string to its unique minimal equivalent element under $\leftrightarrow^*$, by applying string substitution as often as possible, using only the elements in E.

One of our current computer programs for this purpose constructs a finite state automaton W_E for a given finite set of relations E, such that W_E rejects strings if and only if they contain a substring x for some $(x, y) \in E$. It therefore accepts precisely the irreducible elements of A^* under E. It has the additional property that the failure states of W_E are in one-one correspondence with the elements of E, and so when it fails it essentially announces which particular x it has encountered. If E is confluent we can choose $W = W_E$, and then W will be correct in the sense defined above, and we shall have achieved all of our aims.

The automaton W_E can be used to give a fast reduction of a word w, relative to E, as follows. As we read in w, we record the successive states of W_E. A failure means that the left-hand side u of an equation (u, v) in E has been encountered as a substring of w. Using the recorded history, we can go back to the state of W_E just before we started to read u, and replace u in w by v. We then carry on reading w as before.

We illustrate this process when the set E contains only the single equation (bab, aba), corresponding to the rewrite rule $bab \to aba$. Then W_E is shown in Fig. 1 where the start state is S_1, and S_1, S_2 and S_3 are the success states.

Let the input word be $bbab$. Then we have the following successive instantaneous descriptions:

$$S_1bbab,\quad bS_2bab,\quad bbS_2ab,\quad bbaS_3b,\quad bbabS_4,\quad bS_2aba,\quad baS_3ba,$$
$$babS_4a,\quad S_1abaa,\quad aS_1baa,\quad abS_2aa,\quad abaS_3a,\quad abaaS_1,$$

where the word to the left of the state is what has already been read, and the word to the right of the state is what remains to be read.

From a technical viewpoint, it is important to observe that a particular reduction algorithm defines a map $\rho_E: A^* \to A^*$ which maps each string onto a string that is reduced under E and which has the same image in G as the original string. Then E is confluent if and only if ρ_E is independent of the choice of the reduction algorithm. Since E will not usually be confluent in practice, we should decide on a specific algorithm, although it is probably not particularly important which one we choose. To be specific, we define ρ_E as follows. Let $w = w_1 w_2 \ldots w_n \in A^*$. If no substring $w_i \ldots w_j$ of w is the left-hand side of any equation in E, then w is already irreducible, so we set $\rho_E(w) = w$. Otherwise, choose the least j and then the earliest equation in E such that $w_i \ldots w_j$ is the left-hand

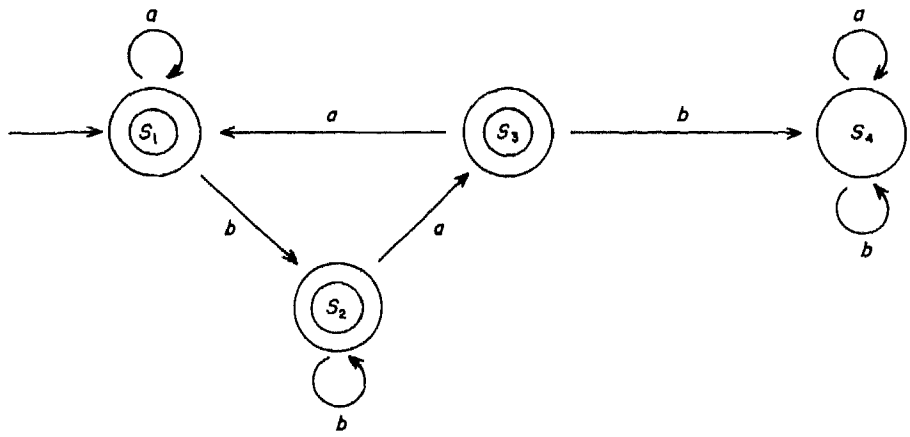

Figure 1.

side of this equation for some i, and replace $w_i \ldots w_j$ in w by the right-hand side of this equation. Repeat this procedure until w has been replaced by an irreducible string w' and define $\rho_E(w) = w'$.

Of course, E will not in general be confluent to begin with. A special case of the *Knuth–Bendix Lemma* tells us that E is confluent if and only if the following conditions are satisfied by each ordered pair $((r, s), (t, u))$ of elements of E.

(i) For all $x, y \in A^*$ with $t = xry$, there exists $w \in A^*$ such that $u \to^* w$ and $xsy \to^* w$.

(ii) For all $x, y, z \in A^*$ with $r = xy$ end $t = yz$, there exists $w \in A^*$ such that $sz \to^* w$ and $xu \to^* w$.

For a proof of this for monoids, see Gilman (1979). The algorithm that we employ attempts to extend E to a confluent set by considering all pairs in E, and then testing the conditions (i) and (ii). If either of these conditions fail then, after using the existing automaton W_E to reduce the relevant words as described above, and after stripping them of any common prefixes and suffixes, we end up with a pair of distinct irreducible words w_1, w_2 which represent the same element in G, and so (assuming $w_1 > w_2$) we adjoin (w_1, w_2) to E. We then have to modify W_E accordingly, which is the trickiest part of the program. Before installing a new equation, we test it for overlaps at each end with the inverse relator equations (aa', ε) and $(a'a, \varepsilon)$. In general, this gives rise to a number of new equations, most of which are redundant. Only the equations which are not redundant are installed. One effect of this is to ensure that the difference of lengths $l(w_1) - l(w_2)$ is equal to 0, 1 or 2, since, if this is greater than two, then the last element of w_1 can be inverted and transferred to the end of w_2, thereby replacing w_1 by a proper prefix of itself.

In some cases, this process will eventually halt with a finite confluent set E, in which case we are very lucky, and we shall have essentially achieved all of our aims, since the program will have proved the correctness of E (and hence also W) for us. This will always occur if G is finite, but that is not very helpful since we are concerned almost exclusively with infinite groups, and in any case the Todd–Coxeter procedure is nearly always far superior for finite groups. It does occasionally occur for infinite groups, particularly for certain Coxeter groups.

Let us assume then that the Knuth–Bendix procedure does not halt, but generates an infinite set E of equations. In one of our implementations, the procedure is interrupted at regular intervals (typically after finding n new equations since the last interruption,

where the user can chose n, with the proviso that E is always closed under overlaps with the inverse relators, as described above, before the interruption). After each such interruption, several calculations are made. Let E_r be the set of equations after the rth interruption. Then the first step is to remove redundant equations from E_r. This is done as follows. For each equation $(v, w) \in E_r$ with $v > w$, we first replace w by its reduced image under the system E_r. We then attempt to reduce v using only the equations in $E_r - \{(v, w)\}$. If v reduces to x, then if $w = x$ we delete (v, w) from E_r, and otherwise we replace it by (w, x) or (x, w) as appropriate. This ensures that the right-hand sides of all equations and all proper substrings of the left-hand sides of all equations in E_r are irreducible under E_r. We shall call E_r *minimal* when it satisfies these conditions. (Note that, when E_r is minimal condition (i) of the Knuth-Bendix Lemma cannot occur.) Of course, some of these equations may later turn out to be redundant or to need modifying in E_s for some $s > r$ but, since an equation can only be removed or modified by the use of shorter equations, any given equation can only be modified finitely many times, and so it will eventually either disappear or remain constant. We can then define the infinite set E of equations as being the set of those equations that lie in E_s for all sufficiently large s. In symbols

$$E = \bigcup_{r=0}^{\infty} \left(\bigcap_{s=r}^{\infty} E_s \right).$$

Then E will be minimal and confluent, and a word $w \in A^*$ will be the minimal representative of $\bar{w} \in G$, if and only if w has no substring equal to the left-hand side of an equation in E. In other words, the map $A^* \to G$ defined by $w \to \bar{w}$ is a bijection when restricted to the elements of A^* that are irreducible under E. Moreover, multiplication in the group can be computed by concatenation of irreducible words followed by reduction.

We shall assume from now on that the group G is known to be shortest word automatic with respect to the given ordering of the generating set A, and so the automata W and M_a exist as in Definition 2.1. Let (v, w) be an equation in E. Then $v = xa$ for some $x \in A^*$, $a \in A$. It follows from the description of E above that w and x are irreducible, and so they lie in $L(W)$. Hence $(x, w)^\dagger \in L(M_a)$, and the word differences of (x, w) are word differences of $L(M_a)$. It follows from Theorem 2.3 that the set of word differences of E (after padding) is finite, and so, for a large enough value of r, all word differences in E will occur as word differences of E_r. We therefore attempt to compute the word differences of E_r at each stage. In the current implementation, the user must make an intelligent guess as to when all word differences of E have been found, and then stop the program.

EXAMPLE 3.1. Let G be the free abelian group

$$\langle a, a', b, b' | aa' = a'a = bb' = b'b = 1,\ ba = ab \rangle.$$

If we order the generators $a < a' < b < b'$, then we obtain the finite confluent set of eight equations mentioned in the Introduction. To make things more difficult, let us use the ordering $a < b < b' < a'$. Then the Knuth-Bendix process does not halt, but generates the infinite set E of equations

$$(a'a, \varepsilon),\quad (aa', \varepsilon),\quad (b'b, \varepsilon),\quad (bb', \varepsilon),\quad (ba, ab),\quad (b'a, ab'),\quad (a'b, ba'),$$
$$(a'b', b'a'),\quad (ab^n a', b^n),\quad (ab'^n a', b'^n),\quad \text{for all } n > 0.$$

The set of word differences of E consists of the finite set

$$\{1, a, b, b', a', ab, ab', ba', b'a'\}$$

(of which we shall number the elements from one to nine, for future reference), and all of these occur already as word differences of the set of 10 equations obtained by only allowing $n = 1$.

There is a fairly serious technical problem that arises at this point. We are not in a position to compute word differences in the way they were defined in Definition 2.2, as elements of G. All that we can compute is a set D_r of strings in A^* that are irreducible under E_r. However, since the set of word differences of E is finite, it follows that, for sufficiently large r, D_r will contain all of the strings of A^* that are irreducible under E and that map onto the word differences of E. It may also contain additional strings since, as we have already seen, not all equations in E_r will eventually lie in E, and so we may get some superfluous word differences. These superfluous word differences will not affect the correctness of the procedure, although they may decrease its efficiency, and they may make it more difficult for us to decide when to stop. Since we shall later be using Theorem 2.5 to prove the correctness of our automata, there is no theoretical need to justify any of these heuristic arguments in the case of successful completion of the whole process.

Let us assume then that we have decided that E_r probably generates all word differences of E. The set $D = D_r$ mentioned above is computed as follows. Let (v, w) be an equation in E_r, where $v = a_1 \ldots a_n$ and $w = b_1 \ldots b_m$, and let $\rho = \rho_{E_r}$ be the string-reduction map defined in section 3. Then we define $s_0 = \varepsilon$ to be the empty string and, for $1 \leq i \leq n$, we put $s_i = \rho(a_i' s_{i-1} b_i)$, where a_i' is the inverse generator of a_i, and b_i is omitted for $i > m$. If it should turn out that s_i is the empty string for some $i < n$, then we are certainly missing a necessary equation from E_r, and so we give up, and carry on with the Knuth-Bendix process (in fact, we have never encountered this phenomenon). We now define D to be the set of all distinct elements s_i coming from all equations in E_r. As pointed out above, we must have $l(v) - l(w) = n - m = 0$, 1 or 2. In fact, we shall assign to each element of D other than the empty string a type, equal to 0, 1 or 2, and replace the set D by the set of typed word differences. (This means that the same element of D could conceivably occur more than once, with different types.) A word difference of type 0, 1 or 2 is one that arises from an equation (v, w) with $l(v) - l(w) = 0$, 1 or 2, respectively, but we do not bother to assign a type to the empty string ε, which is always the first element of D.

We now output what we shall call the *principal transition table.* Strictly speaking, this is a set of quadruples in $A \times A^{\dagger} \times D \times D$. In fact we shall write it as a mapping $\varphi : A \times A^{\dagger} \times D \to D$, although φ is neither a complete function, nor even single-valued. For each string s_{i-1} with $i > 0$ arising from the equation (v, w) as above, we insert an entry $\varphi(a_i, b_i, s_{i-1}) = s_i$ in the transition table, where of course $b_i = \$$ if $i > m$. Owing to the fact that we have not assigned a type to ε, φ may be multi-valued on (a, b, ε). Since the transition table consists of a series of entries, however, this point is not important.

Returning to Example 3.1, and letting D be the set of nine word differences (numbered 1-9) described there, we find that word-differences 2, 3, 4, 5, 8 and 9 have type 2, whereas 6 and 7 have type 0. The principal transition table contains the following entries:

$(a, b, 1) \to 8$, $(a, b', 1) \to 9$, $(a, \$, 1) \to 5$, $(b, a, 1) \to 7$, $(b, \$, 1) \to 4$, $(b', a, 1) \to 6$,

$(b', \$, 1) \to 3$, $(a', b, 1) \to 6$, $(a', b', 1) \to 7$, $(a', \$, 1) \to 2$, $(a, \$, 2) \to 1$, $(b, \$, 3) \to 1$,

$(b', \$, 4) \to 1$, $(a', \$, 5) \to 1$, $(a, b', 6) \to 1$, $(b, a', 6) \to 1$, $(a, b, 7) \to 1$, $(b', a', 7) \to 1$,

$(b, b, 8) \to 8$, $(b, \$, 8) \to 5$, $(b', b', 9) \to 9$, $(b', \$, 9) \to 5$,

We shall say that the principal transition table is *correct* if the following conditions hold. For each equation (v, w) of E, where $v = a_1 \ldots a_n$ and $w = b_1 \ldots b_m$, and each i with $1 \leq i \leq n$, D contains the (unique) element $s_i \in A^*$ with the type $n - m$, such that s_i is irreducible under E and $\bar{s}_i = (\bar{a}_1 \bar{a}_2 \ldots \bar{a}_i)^{-1}(\bar{b}_1 \bar{b}_2 \ldots \bar{b}_i)$. Furthermore, each transition $\varphi(a_i, b_i, s_{i-1}) = s_i$ must be present in the table. Some additional elements of D or some additional transitions may also be present in the table, but this does not affect the correctness.

The following lemma contains some properties of the principal transition table as constructed, which we shall need in the next section. Of course, we do not know at this stage whether or not the table is correct.

LEMMA 3.2. (i) *If $\varphi(a, b, s)$ is defined and $s \neq \varepsilon \neq \varphi(a, b, s)$, then $\varphi(a, b, s)$ has the same type as s.*

(ii) *If $\varphi(a, b, s) = \varepsilon$ and s has non-zero type, then $b = \$$.*

(iii) *If $\varphi(a, b, \varepsilon) = s$ and s has type 0, then $b < a$ in the ordering of A.*

PROOF. (i) is clear from the definition of φ. If (ii) were false then, for some equation $(v, w) = (a_1 \ldots a_n, b_1 \ldots b_m)$ of E_r with $m < n$, one of the strings s_i defined above with $i < n$ would reduce to ε under E_r. However, as we mentioned above, we have specifically ensured that this does not occur. For the same reason, if $\varphi(a, b, \varepsilon) = s$ and s has type 0, then there is an equation $(a_1 \ldots a_n, b_1 \ldots b_m)$ of E_r with $n = m$, where $a = a_1$ and $b = b_1$. Since equations are always stripped of common prefixes before they are installed, we must have $a \neq b$, and so $b < a$.

We shall describe in the next section how the principal transition table is used to construct the automaton W. The automata M_a will be constructed according to the recipe described in Corollary 2.4. For this purpose, we require the set D', which we define to be the union of the word differences of the $L(M_a)$, and which will usually be larger than D. (The set D of Corollary 2.4 is represented by the set D' here.) Given an element $(v, w)^\dagger \in L(M_a)$, v and w will of course be irreducible in E_r, but some proper suffix of va may be reducible, and so the equation (va, w) will not necessarily occur in E_r; the reduction $va \to^* w$ may involve several applications of equations in E_r. In order to compute a sensible candidate for D', we therefore have to extend E_r to a larger set E_r' of equations of the form (va, w), where $a \in A$ and v and w are irreducible under E_r'; we may also assume that v and w have no common prefix.

To do this, we start with $E_r' = E_r$. Now, for all equations (t, u) in E_r' and (v, w) in E_r, we look for overlaps between t and w of the form $t = xy$ and $w = yz$ with $x, y, z \in A^*$ and $y \neq \varepsilon$. Then (xv, uz) is a candidate for inclusion in E_r'. Before inserting it, we write $xv = xv'a$, with $a \in A$ and replace xv' and uz by their irreducible images under E_r (see Figure 2). We then strip any common prefix from the left- and right-hand sides of the new equation and, to avoid repetition, we check that this equation does not already lie in E_r'.

Of course, like the Knuth–Bendix algorithm itself, this procedure will in general continue indefinitely. We therefore interrupt it at regular intervals, and compute the set D' of word differences of the words (v, w), where (va, w) in E_r' and $a \in A$. As was the case for D, we actually compute a set of strings that are irreducible under E_r rather than a set of group elements, and we compute them in the same way as we described above for D. If and when this set appears to have become constant, we stop, and use it as our candidate for D'. Since the word differences of $L(M_a)$ will be the inverses of those of $L(M_{a^{-1}})$ (where

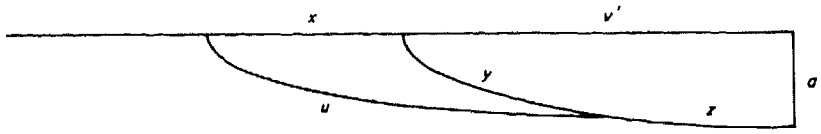

Figure 2.

a' is the inverse of the generator a), we always ensure that D' is closed under inversion in the obvious sense as applied to strings. In contrast to the situation for D, we do not bother to assign types to the elements of D'.

We now output what we shall call the *product transition table*, which will be used to construct the automata M_a. This is similar to the principal transition table, but is output as a (total) function $A^\dagger \times A^\dagger \times D' \to D' \cup \{0\}$ and, for $s \in D'$, (a, b, s) is mapped to $\rho(a_i' s b_i)$ if this lies in D', and to 0 otherwise. (Some of the minor details of these processes, such as assigning types to elements of D but not to those of D', and to differences in the definitions of the transition tables may seem arbitrary to the reader. Usually, they have resulted from experiment and experience rather than from logical considerations. Indeed, the authors have, to some extent independently, tried several different implementations of these ideas, and we are only describing one of these here.)

In Example 3.1, the sets E_r' are larger than E_r, and contain equations $(b^n a, ab^n)$ for $n > 0$, for example. However, no new word differences arise. The product transition table will of course contain many more entries than the principal transition table.

This completes the description of our implementation of the Knuth–Bendix algorithm. Of the output, only the two transition tables will be required for the construction of the automata.

4. The Construction of the Finite State Automata

We shall now describe how the principal transition table that was output by our Knuth–Bendix algorithm is used to construct a candidate for our finite state automaton W, and for the associated word reduction procedure.

The set of states of W will consist of the set of all subsets of $D - \{\varepsilon\}$ (where ε is the empty string), together with a failure state 0. The initial state is the empty subset, and all states other than 0 are success states. The state 0 is mapped onto 0 by any input. Otherwise, let S be a subset of $D - \{\varepsilon\}$ and let $a \in A$. Then the image S^a of S under a is defined as follows. Let $S' = S \cup \{\varepsilon\}$, let $\varphi : A \times A^\dagger \times D \to D$ be the "mapping" defined by the principal transition table, and let

$$T = \bigcup_{s \in S'} \bigcup_{b} \varphi(a, b, s),$$

where the second union is taken over all $b \in A^\dagger$ for which $\varphi(a, b, s)$ is defined. Then $S^a = T$ if $\varepsilon \notin T$ and otherwise $S^a = 0$. This definition of W is justified by the following theorem.

THEOREM 4.1. *The finite state automaton W defined above is correct if the principal transition table is correct.* (*Correctness was defined precisely in section* 3.)

PROOF. Suppose that the principal transition table is correct. Let $w = a_1 a_2 \ldots a_n \in A^*$, and suppose that W is in state S_i after reading the ith character a_i of w. We have to show

that W rejects w if and only if w is reducible under the infinite confluent set of equations E. Suppose first that w is reducible. Then some substring of w is the left-hand side of a padded equation $(a_{l+1}\ldots a_m, b_{l+1}\ldots b_m)$ of E (where b_m and b_{m-1} may be equal to \$). Let $g_i=(\bar{a}_{l+1}\ldots\bar{a}_i)^{-1}(\bar{b}_{l+1}\ldots\bar{b}_i)$ for $l\leq i\leq m$. (Note that $g_l=g_m=1$.) Then, since the principal transition table is correct, for each such g_i, the unique string $s_i\in A^*$ (with the correct type) that is irreducible under E and maps onto g_i lies in D, and each transition $\varphi(a_i, b_i, s_{i-1})=s_i$ for $l\leq i\leq m$ occurs in the table. If $S_i=0$ for some $i<m$, then W certainly rejects w, so suppose not. Then the transition $\varphi(a_{l+1}, b_{l+1}, \varepsilon)=s_{l+1}$ ensures that $s_{l+1}\in S_{l+1}$, and similarly, by induction, we get $s_i\in S_i$ for $l<i<m$. Finally, the transition $\varphi(a_m, b_m, s_{m-1})=\varepsilon$ makes $S_m=0$, and so w is rejected by W.

Suppose conversely that w is rejected by W and let m be minimal such that $S_m=0$. Let $w_i=a_1\ldots a_i$ for $0\leq i\leq m$. For each $s\in S_i$ with $0\leq i<m$, we shall associate a string $v=v(i, s)$ of A^* with $\bar{w}_i^{-1}\bar{v}=\bar{s}$. Furthermore, if s has type 0, then we shall also require $v<w_i$. We define the $v(i, s)$ inductively on i. Since S_0 is empty, there is nothing to define for $i=0$, so suppose that the $v(j, s')$ have been defined for all S_j with $j<i$ and all $s'\in S_j$. Let $s\in S_i$. Then, for some $s'\in S_{i-1}\cup\{\varepsilon\}$ and some $b\in A^\dagger$, we have $\varphi(a_i, b, s')=s$. If $s'\neq\varepsilon$ then s' and s have the same type, by Lemma 3.2(i). Put $v'=w_{i-1}$ if $s'=\varepsilon$, and otherwise let $v'=v(i-1, s')$ be the word associated with s'. If s' is not unique, then we select s' such that v' is minimal under $<$. Then the word $v=v(i, s)$ associated with s is defined by $v=v'b$ if $b\neq\$$, and $v=v'$ if $b=\$$. We see from the definition of φ that this satisfies $\bar{w}_i^{-1}\bar{v}=\bar{s}$ (see Figure 3). Suppose that s has type 0. Then, if $s'\neq\varepsilon$, we have $v'<w_{i-1}$ by inductive assumption, and so clearly $v=v'b<w_i$. On the other hand, if $s'=\varepsilon$, then $v=w_{i-1}b<w_{i-1}a=w_i$ follows from Lemma 3.2(iii). Thus we have defined the $v(i, s)$ as specified. Now since $S_m=0$, for some $s\in S_{m-1}$ and some $b\in A^\dagger$, we have $\varphi(a_m, b, s)=\varepsilon$. Let $v=v(m-1, s)$ be the word associated with s. Then, since $\bar{w}_{m-1}^{-1}\bar{v}=\bar{s}$, we have $\bar{w}_m=\bar{v}\bar{b}$. If s has non-zero type, then $b=\{\$\}$ by Lemma 3.2(ii), and so $v<w_m$ since v is shorter than w_m. On the other hand, if s has type 0, then $v<w_{m-1}$ by assumption, and so $vb<w_m$. In either case, w_m is not the minimal element under $<$ that maps onto $\bar{w}_m$.

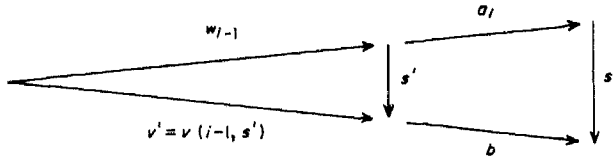

Figure 3.

The proof of the above theorem includes the recipe for a fairly efficient algorithm for reducing an arbitrary string in W to normal form. For this purpose, we would need to compute and remember the words $v(i, s)$ that are associated with the elements s of the states S_i as described in the proof. Then, whenever W rejects a word w, we can immediately substitute v or vb for the prefix w_m of w and restart. Since we are replacing w by a smaller element under $<$ that maps onto the same element of G, this process will eventually result in a word $x\in L(W)$ with $\bar{x}=\bar{w}$. Then, if W is correct, x will be in normal form.

In fact it is impractical to construct W completely, because D will have far too many subsets, but fortunately this is not necessary. We only need to compute the action of the generators on the *accessible* states of W. To do this, let $\mathcal{S}=\{S_1\}$, where S_1 is the initial state (the empty subset). For each $S_i\in\mathcal{S}$, we then compute the action of all generators

on S_i, and every time that we get a state S_n that is not already in $\mathcal{S}$, we simply adjoin it to $\mathcal{S}$. Eventually this process must stop, and then $\mathcal{S}$ will consist of the set of accessible states. In the examples that we have run, $\mathcal{S}$ has usually been reasonably small (some examples are given in section 6).

Returning to Example 3.1, and using the same numbering for the elements of D, we find that there are just seven accessible states of W, namely $\{\}$, $\{5, 8, 9\}$, $\{4, 7\}$, $\{3, 6\}$, $\{2, 6, 7\}$, $\{4, 7, 5, 8\}$, and $\{3, 5, 6, 9\}$, and if we number these 1-7, then W has the following transition table.

	a	b	b'	a'
0	0	0	0	0
1	2	3	4	5
2	2	6	7	0
3	0	3	0	5
4	0	0	4	5
5	0	0	0	5
6	0	6	0	0
7	0	0	7	0

and it is easy to see directly that this automaton does indeed accept precisely those words that are in normal form for this group.

Given the automaton W and the product transition table based on the larger difference set D', the product automata M_a can be constructed exactly as described in Corollary 2.4 (using D' in place of D), except that again we need only construct the action of the generators on the accessible states. Before doing this, however, it is important to minimize W; that is, to replace W by an equivalent automaton with the smallest possible number of states. It is even more important to minimize the M_a before proceeding to the relation verification stage to be described in the next section. There is a standard algorithm for minimizing the number of states of a finite state automaton, which is described in Chapter 4.13 of Aho *et al.* (1974). In Example 3.1, for example, the automata M_a, M_b, $M_{b'}$ and $M_{a'}$ initially have 59 states, but they have only 8, 9, 9 and 8 states respectively, after minimization.

We conclude this section by observing that our automata W and M_a satisfy hypotheses (i)-(iii) of Theorem 2.5. Of these (i) and (ii) are immediately clear from the construction of the M_a as described in Corollary 2.4. Since W has only one failure state, and that is fixed by all generators, it is clear that $L(W)$ is non-empty (since it accepts the empty string), and any prefix of a word in $L(W)$ is also in $L(W)$. Furthermore, since the equations (aa', ε) and $(a'a, \varepsilon)$ all lie in E (or else (a, ε) and (a', ε) lie in E), D' must contain the elements $\{\hat{a}\}$ for all $a \in A$, where $\hat{a} = \rho(a)$ is the reduction under E_r of the string a. The product transition table will therefore contain entries $(\$, a, \varepsilon) \to \hat{a}$ for all $a \in A$. It is now clear that (iii) holds also.

5. Verifying the Relations

In order to complete the proof of the correctness of the W and the M_a, we need to verify hypothesis (iv) of Theorem 2.5. We shall describe how to do this in this section. There is only one new idea involved. Let M_1 and M_2 be two finite state automata with input alphabet $A^\dagger \times A^\dagger$ which accept only padded words over $A \times A$. Then the *composite* $M_{1,2}$ of M_1 and M_2 is defined as follows. $M_{1,2}$ will also have input alphabet $A^\dagger \times A^\dagger$ and accept only padded words over $A \times A$ and, for $u, v \in A^*$, we have $(u, v)^\dagger \in L(M_{1,2})$ if and only if there exists $w \in A^*$ with $(u, w)^\dagger \in L(M_1)$ and $(w, v)^\dagger \in L(M_2)$.

In order to verify hypothesis (iv) of Theorem 2.5, we can proceed as follows. For each defining relator $r = a_1 a_2 \ldots a_n$ of G, we use our candidates for M_{a_i} that we have already constructed to define successively the composite automata $M_{a_1 a_2}, M_{a_1 a_2 a_3}, \ldots, M_{a_1 a_2 \ldots a_n} = M_r$. Then the hypothesis will be satisfied for this relation if and only if $(u, v)^\dagger \in L(M_r) \Leftrightarrow u = v \in L(W)$. Checking this final equivalence is easy in practice, since it amounts to checking that M_r is equivalent to the diagonal automaton M_S derived from W. But after minimization, two automata are equivalent if and only if there is a bijection between their sets of states which is preserved by their state-transition tables, and this is readily verified. Indeed, in our implementation, this can usually be done simply by checking whether two files are identical.

It remains only to describe how the composite of two automata can be constructed. Let the sets of states of M_1 and M_2 be S_1 and S_2, respectively, with initial states s_1^0 and s_2^0. The states of $M_{1,2}$ are the subsets of $S_1 \times S_2$, with initial state $\{(s_1^0, s_2^0)\}$. Let $\mathcal{S}$ be a subset of $S_1 \times S_2$ and $(a, b) \in A^\dagger \times A^\dagger$. Then the image $\mathcal{S}^{(a,b)}$ of $\mathcal{S}$ under (a, b) is defined by

$$\mathcal{S}^{(a,b)} = \bigcup_{(t_1,t_2)\in\mathcal{S}} \bigcup_{c\in A^\dagger} (t_1^{(a,c)}, t_2^{(c,b)}).$$

Let S be the set consisting of those states of $M_{1,2}$ that contain an element (s_1, s_2), where s_1 and s_2 are success states of M_1 and M_2 respectively. S is then roughly equal to the set of success states of $M_{1,2}$, but we have not dealt with the padding problem. It may happen that, for certain $u, v \in A^*$ for which $(u, v)^\dagger$ ought to be accepted, we have to apply $(u, v)^\dagger$ followed by a number of copies of $(\$, \$)$ in order to arrive at a state in S. So, for each state s of $M_{1,2}$, we check whether the repeated application of the input $(\$, \$)$ to s can lead to a state in S. If so, then we add s to S, and redefine $s^{(\$,\$)}$ to be a dead state (that is, a failure state that is mapped to itself by any input). After this modification, S is equal to the set of success states of $M_{1,2}$.

Once again, we cannot and need not construct the complete transition table for $M_{1,2}$, but it is only necessary to do this for the accessible states. After each such construction, it is essential to minimize the resulting composite automaton before proceeding further.

Normally, we carry out this process first on the relators aa' and $a'a$, where a' is the inverse of the generator a'. From the proof of Theorem 2.5, we observe that, if this is successful, then the automata W and M_a certainly correspond to a group, although this could in principle still be a larger group than G. Checking the hypothesis for a relator $a_1 a_2 \ldots a_n$ is now equivalent to checking that the composite automata $M_{a_1 a_2 \ldots a_m}$ and $M_{a'_n a'_{n-1} \ldots a'_{m+1}}$ are equivalent, where we can choose a value of m with $1 \leq m \leq n$, and it is usually most efficient to do this with m equal to about $n/2$. In practice, other short cuts are possible. For example, if a word $a_1 a_2 \ldots a_m$ occurs as a subword of several different relators, then the appropriate composite $M_{a_1 a_2 \ldots a_m}$ can be used each time. Often, we have relators (or subwords of relators) of the form $(a_1 a_2 \ldots a_m)^n$. In this case, it is most efficient to start by constructing $M_{(a_1 a_2 \ldots a_m)^r}$ for $r = 1, 2, 4, \ldots$.

6. Some Remarks on Implementation and Some Performance Statistics

All programs were written in the *C* language and were run on a Sun 3/60 workstation. Our implementation of the Knuth-Bendix algorithm is simple and straightforward. As we have already mentioned, all new equations are tested for overlaps (at both ends) with the inverse relations $(aa', 1)$ as soon as they are found. This ensures that, after removing

redundant equations, the lengths of the left- and right-hand sides of all equations differ by at most two. It might also be efficient to test new equations immediately against other very short equations (such as the defining relations), but we have not experimented with this idea. What we have done is to give higher priority in the process queue for those equations that give rise to new word differences. In many examples, this results in the complete set of word differences being found much more quickly. Owing to the simplicity of implementation, the algorithm runs very quickly for small sets of equations E (up to a few hundred), but then slows down rapidly. Its practical limit is about 10 000 equations, which would take several days of cpu-time to generate. This seems to be a time problem rather than a space problem, and there is potential for an improvement in performance by using more sophisticated string searching techniques to look for overlaps in the equations. In fact we have implemented such an improvement, which we hope will ultimately be able to handle much larger sets of equations, since there are substantial savings in time, and probably simultaneous savings in space.

The construction of W does not seem to present any particular problems, since both the number of its states and the size of its input alphabet are usually reasonably small in comparison with other data structures being handled. However, the M_a and the associated composites are more problematical. Particularly before minimization, they tend to have many more states than W, and their input alphabet has the much larger size $(|A|+1)^2$.

We have experimented with two methods of storing the transition tables of finite state automata. The first, which we call the *vector* representation, is to represent the transitions from each state s by a vector $(s_1, s_2, \ldots, s_n)$, giving the transitions of each element of the input alphabet acting on s. In the second, the *string* representation, these transitions are represented as a string of the form $(r, a_1, s_1, a_2, s_2, \ldots, a_r, s_r)$, where a_i maps state s to state s_i and all other elements of the input alphabet maps s to a dead state. The vector representation tends to be more efficient for subsequent computation, since the transitions can be accessed immediately, and we have used this form for W. Since the product and the composite automata have much larger input alphabets, and their transition tables are very sparse (the vast majority of their states will only accept one or two elements of $A^\dagger \times A^\dagger$ as input), it seems necessary to use the string representation form for them. Even so, in complicated examples, it is the size of these composite automata before minimization that is currently the largest obstacle to further progress, and unless some technique for incremental minimization can be developed, it will be necessary to make greater use of secondary storage, which will of course result in a large increase in process times.

We conclude with some performance statistics on some selected examples. To save space, we shall omit the inverse generators a' and relators aa' and $a'a$ in our group definitions but, unless there is a relator aa, they are tacitly understood to be present in the presentations.

EXAMPLE 6.1. $G = \langle a, b | bab = aba \rangle$ (the trefoil knot group). After the Knuth-Bendix procedure had generated about 30 equations, the difference tables became constant. The set D contained 14 (typed) word differences, and the larger set D' (as defined in section 4) contained 20 (untyped) word differences. The automaton W initially had 20 states, which minimized to 16, whereas M_a, $M_{a'}$, M_b, $M_{b'}$ initially had 308 states, and minimized to 37, 37, 43 and 43, respectively. The composite automaton $M_{aa'}$ initially had 177 states and minimized to 16 (and was then equal to the diagonal $M_\$$ of W), with similar numbers of $M_{a'a}$, $M_{bb'}$ and $M_{b'b}$. We then defined M_{ba} (217 states minimizing to 48) and M_{bab}

(243 states minimizing to 45) and similarly for M_{ab} and M_{aba}. M_{bab} and M_{aba} turned out to be identical, which completed the correctness proof. The complete computation took only a few seconds cpu-time.

EXAMPLE 6.2. $G = \langle a, b | a^n = b^3 = (ab)^2 = 1 \rangle$ (the von Dyck group $D(2, 3, n)$). This was also easy for small values of n (we tried it for $n = 7, 8, 9, 10, 11$), and the complexity was increasing slowly (probably linearly) with n. The sizes involved were similar to those in Example 6.1.

EXAMPLE 6.3. $G = \langle a, b, c, d | a^{-1}b^{-1}ab = c^{-1}d^{-1}cd \rangle$ (the genus 2 surface group). If we use the non-obvious ordering of the generators $a < a' < d < d' < b < b' < c < c'$ we in fact get a finite confluent set of equations, which yields W with 25 states after minimization. Indeed, it is shown in Le Chenadec (1986) that there are finite confluent sets of equations for all of the 2-dimensional surface groups, provided that the generators are ordered correctly. With the default ordering $a < a' < b < b' < c < c' < d < d'$, we get an infinite set of equations. The sets D and D' had sizes 41 and 32, respectively. The automaton W initially had 53 states, which minimized to 48, whereas M_a, etc. initially had 428 states, and minimized to about 73, depending on the generator. The largest composite automaton that had to be defined was $M_{a'b'ab}$ and this had 304 states minimizing to 143. This example took a total of about 80 seconds cpu-time to run.

The above examples are all sufficiently transparent that it would not be impossible to write down the complete infinite system E as a regular set of equations and verify confluence. In the next example, that would be considerably more difficult, since the set E_r has to grow much larger before its word differences become constant. This is in fact a 4-generator hyperbolic Coxeter group, and several other such examples behave similarly.

EXAMPLE 6.4.

$$G = \langle a, b, c, d | a^2 = b^2 = c^2 = d^2 = (ab)^2 = (ac)^2 = (ad)^5 = (bc)^5 = (bd)^3 = (cd)^2 = 1 \rangle.$$

We ran the Knuth-Bendix procedure for about 3200 seconds, after which E_r contained about 500 equations and the sets of word differences of D and D' appeared to have become constant with sizes 76 and 87, respectively. The procedure to make the automata took 80 seconds. W had 186 states and minimized to 126, whereas M_a, M_b, M_c and M_d had 4219 states initially and minimized to about 550. Forming the composite automaton for verifying the relators took a total of about 1360 seconds. The largest of these M_{adadad} had 11 738 states initially and reduced to 957.

Example 6.4 is by no means trivial, but it is still a relatively easy computation. Unfortunately, we have not yet been able to complete the computation for any significantly more complex examples. We are currently attempting this with a 5-generator hyperbolic Coxeter group. The first problem here is that it becomes more difficult to know when to interrupt the Knuth-Bendix procedure, since it will sometimes produce several hundred new equations before finding one with a new word difference. Our current candidate for E_r has 434 word differences. We used this to produce W with about 50 000 states minimizing to 1190, and the M_a had about 250 000 states minimizing to about 28 000. Currently, we have not had enough space available to generate even the first composite M_{aa}. It is interesting that time seems to be the principal constraint for the Knuth-Bendix procedure, whereas space is the obstacle to forming the composite automata.

References

Aho, A. V., Hopcroft, J. E., Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA.

Benninghofen, B., Kemmerich, S., Richter, M. M. (1987). Systems of reductions. *Lecture Notes in Computer Science* **277**.

Cannon, J. W. (1984). The combinatorial structure of discrete hyperbolic groups. *Geometriae Dedicata* **16**, 123–148.

Cannon, J. W., Epstein, D. B. A., Holt, D. F., Paterson, M. S., Thurston, W. P. (1990). Word processing and group theory. Preprint (from Epstein or Holt).

Gilman, R. H. (1979). Presentations of groups and monoids. *J. Alg.* **57**, 544–554.

Gilman, R. H. (1985). Enumerating infinitely many cosets. In: (Atkinson, M. D. ed.) *Computational Group Theory*, pp. 51–55. London: Academic Press.

Gromov, M. (1987). Hyperbolic groups. In: (Gersten, S. M. ed.) *Essays in Group Theory* Springer-Verlag, *Mathematical Sciences Research Institute* **8**, 75–263.

Hopcroft, J. E., Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA.

Le Chenadec, P. (1986). A catalogue of complete group presentations. *J. Symbolic Computation* **2**, 363–381.